Slide 1

Administrivia

- (None?)

Slide 2

Basic OpenMP Constructs, Etc. — Overview

- `#pragma omp parallel` before a block launches a "team" of threads, which continue until the end of the block. Code after the block executes only after all threads have completed the block.
- `#pragma omp master` or `#pragma omp single` within a parallel block says only one thread will do following block.
- `#pragma omp for` (within parallel block) says iterations of the following `for` loop are split among threads. Sort of the workhorse construct for OpenMP; many options.
- Several synchronization-related keywords.
- Several library functions.

Slide 3

Basic OpenMP Constructs — Parallel `for`

- By default, variables are shared, and semantics of initial, final values are a little complicated.
- `private` can be used to give each thread its own copy of a variable.
- `reduction` can be used to give each thread its own copy of a variable and have them combined ("reduced") at end.
- `schedule` lets you choose how iterations are split among threads — statically/evenly or at runtime.

Slide 4

Example — Numerical Integration

- Compute π by integrating $\int_0^1 \frac{4}{1+x^2}\,dx$.
- Do this numerically by approximating area under curve by many small rectangles, computing their area, adding results.
- Sequential program fairly straightforward. (`num-int-seq.c` on "sample programs" page).
- How to "parallelize'?

Slide 5

Parallel Version of Numerical Integration — Strategy

- Basic strategy seems sort of obvious? most of the processing consists of adding up items computed in a `for` loop, so "parallelize" that: Parcel out iterations of loop among threads, have each thread compute a partial sum, and then combine partial sums.
- But it seems like there might be some issues: How to split iterations among threads? What about shared variables (here, `x` and `sum`)? Probably need to do *something*, no?

Slide 6

Parallel Version of Numerical Integration — Code

- (See example code.)

Slide 7

More OpenMP — Synchronization Constructs

- `critical` — only one thread at a time executes this block of code. (Example — `synch-2.c` on sample programs page.)
- `barrier` — threads wait here until all have arrived. Implicit barrier at end of parallel region.
- `single` — only one thread executes this block.
- Several others — `atomic`, `flush`, `ordered`, `master`. More about them in the specification.

Slide 8

More OpenMP — Locks

- `omp_lock_t` — declares a lock variable.
- `omp_init_lock`, `omp_destroy_lock` — create and destroy.
- `omp_set_lock` — acquire lock (wait if necessary).
- `omp_unset_lock` — release lock.
- Other functions described in specification.
- Example — `synch-3.c` on sample programs page.

Slide 9

Measuring Performance

- Absolute performance what most end users want, and they have large problems that run for a while.
- For this course, however, I say it makes sense to focus on small problems so we can do more tests / measurements — just not *too* small (say, under 10 seconds).

  I also like to time just the computational part of the program.
- Note also that with remote login you have a choice as to where to run programs, so when you get ready to measure performance, might want to log onto one of the "server" machines. Dione is older and slower but has a lot of PEs!

Slide 10

Homework 1 — Overview

- Assignment asks you to parallelize a sequential program fairly similar to numerical integration example:

  The sequential program estimates the value of π by simulating throwing "darts" at a square board and counting how many fall within an inscribed quarter-circle.

  If the board is a square of size 1, its area is 1, and the area of the quarter-circle is $\pi/4$. If the darts are thrown randomly, and there are enough of them, dividing the number that fall inside the quarter-circls by the total number should give an approximation to $\pi/4$.
- The assignment will eventually ask you to do this in each of the programming environments we'll use, as a way of getting started with them. We'll do it twice, once just to get started and to discover some possibly-subtle pitfalls, and again to address those pitfalls.

Slide 11

Homework 1, Continued

- Details coming soon; I can e-mail you sequential code right away if you're eager to get started?